LAB 7

Objective: Learn how to use volumes inside Pods. Learn how to create secrets and ConfigMaps and access them inside Pods.

This lab has three independent parts:

- Mount a volume in two containers via a Pod manifest
- Create a secret and use it to run a MySQL Pod
- Create a ConfigMap and mount it inside a Pod.

Using Volumes

Let's get started with volumes. You can do this part using the description below or skip to the step-by-step instructions.

- Write a single Pod manifest which contains two containers and one volume. Mount the volume in different paths in each of the containers and use **`kubectl exec`** to touch a file. Read the file from the other container. This will show you how to share data between containers in a Pod using a volume.

Let's create a Pod using the provided manifest **`volumes.yaml`** (retrieve it from *Course Resources*). It starts one Pod with two containers in it, one called **`busy`** and the other one called **`box`**.

```
$ kubectl create -f volumes.yaml
$ kubectl get pods
NAME      READY     STATUS    RESTARTS   AGE
vol       2/2       Running   0          22s
```


Connect inside the **busy** container, touch a file in the **busy** directory. Then, connect to the **box** container and see that same file in the **box** directory.

```
$ kubectl exec -ti vol -c busy -- touch /busy/lfs248
$ kubectl exec -ti vol -c box -- ls -l /box
total 0
-rw-r--r--    1 root     root             0 Dec 20 13:28 lfs248
```

A volume is being shared between the two containers in the Pod. The volume is mounted in different paths. Indeed, if you check the manifest **volumes.yaml** you see a *volumes* section common to the Pod, and *volumeMounts* sections specific to each container in the Pod. This is how volumes are accessed in containers.

```
...
spec:
  containers:
  - image: busybox
...
    volumeMounts:
    - mountPath: /busy
      name: test
    name: busy
  - image: busybox
...
    volumeMounts:
    - mountPath: /box
      name: test
    name: box
  volumes:
  - name: test
    emptyDir: {}
```


Using Secrets

- Launch a MySQL pod using a secret to store the MySQL root password. The MySQL image does not run without specifying this password. Create a secret with **`kubectl`** and then read this secret inside the Pod using an environment variable.

The Docker Hub official MySQL image needs to have a **`MYSQL_ROOT_PASSWORD`** environment variable set to run. In a Pod manifest, this looks like the snippet below:

```
spec:
  containers:
  - image: mysql:5.5
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: root
```

However, this is sub-optimal, as it shows the value of the password in the actual manifest. It would be better to create a secret and bind that secret to the Pod at runtime. We can do this. First create a secret by hand with **`kubectl`**:

```
$ kubectl create secret generic mysql --from-literal=password=root
```

You can then use this secret inside a Pod like so:

```
spec:
  containers:
  - image: mysql:5.5
    env:
    - name: MYSQL_ROOT_PASSWORD
      valueFrom:
        secretKeyRef:
          name: mysql
          key: password
```


Use the manifest provided (in the *Course Resources*) to create the MySQL Pod and enter the container to check that the database is running:

```
$ kubectl exec -ti mysql -- mysql -uroot -proot
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.5.54 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights
reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

mysql>
```

Using ConfigMaps

- Create a ConfigMap with **`kubectl`** and mount it inside a Pod. Then, get inside the container and verify that the file is there.

```
$ kubectl create configmap map --from-file=configmap.md
configmap "map" created
$ kubectl get configmaps
NAME      DATA      AGE
map       1         5s
```


Use the **configmap.yaml** file (retrieve it from *Course Resources*) to create the Pod. If you check the content of the **configmap.md** file (retrieve it from *Course Resources*) inside the container, you will see that it corresponds to the original file:

```
$ kubectl create -f configmap.yaml
$ kubectl exec -ti configmap-test -- ls /config
configmap.md
```

Using the ConfigMaps resource, we can share files between containers and load configurations inside containers.

A ConfigMap can be mounted as a volume inside a Pod, just like a regular volume. There are additional ways to refer to a ConfigMap, especially using environment variables.

```
...
spec:
  containers:
  - image: busybox
...
    volumeMounts:
    - mountPath: /config
      name: map
    name: busy
  volumes:
    - name: map
      configMap:
        name: map
```

